Amendments to the Specification:

Please replace the paragraph at page 1, lines 10-13 with the following amended paragraph:

Many [[Java]] JAVA® technologies currently require the generation of classes based on some input class. A prime example is ~~Enterprise Java Beans (EJBs)~~ ENTERPRISE JAVA BEANS® (EJBs®), which require the generation of a particular application server's container classes to wrap the developer supplied bean class.

Please replace the paragraph at page 3, lines 8-16 with the following amended paragraph:

In one embodiment of the present invention, a method of generating code to be deployed in an application server comprises the steps of receiving a compiled file to be deployed, introspecting this input class to generate information relating to the input class, generating a ~~markup language~~ markup language description of the input class based on the generated information relating to the input class, creating an event handler for a method node found in the ~~markup language~~ markup language description, registering the event handler, parsing the ~~markup language~~ markup language description and invoking the registered event handler, and generating output code using the invoked event handler.

Please replace the paragraph at page 3, line 17 to page 4, line 12 with the following amended paragraph:

In one aspect of the present invention, the archive file is a compiled ~~Enterprise Java Bean~~ ENTERPRISE JAVA BEAN® (EJB®) file. The step of introspecting an input

class included in the archive file may comprise the steps of extracting information identifying methods included in the input class and for each method, extracting information relating to parameters of the method and exceptions generated by the method. The step of generating a ~~markup language~~ markup language description of the input class may comprise the step of generating an Extensible ~~Markup Language~~ Markup Language description of the input class based on the generated information relating to the input class. The step of creating an event handler may comprise the step of creating a programming language implementation of a Simple Application Programming Interface for Extensible Markup Language (SAX) event handler for a method node found in the class' XML description. The step of parsing the XML description and invoking the registered event handlers may comprise the step of parsing the markup language description using a SAX parser and invoking the registered SAX event handlers.

Please replace the paragraph at page 5, line 18 to page 6, line 7 with the following amended paragraph:

The present invention is a system and method for generating code that reduces the cost for large projects and that provides easier and quicker ways of finding errors and making modifications. In a preferred embodiment, the present invention relies on the Simple API for XML (SAX) to drive the generation of code. SAX is an event-driven, serial-access mechanism for accessing XML documents. SAX was the first widely adopted API for XML in [[Java]] JAVA®, and is a "de facto" standard. A typical version

of SAX contains a number of core classes and interfaces together with optional helper classes and demonstration classes. SAX is described in greater detail below.

Please replace the paragraph at page 6, line 15 to page 7, line 6 with the following amended paragraph:

An Application Server generally interacts with both clients 102 and back-end ~~EIS systems~~ Enterprise Information Systems (EIS) 104. The application server provides an ~~Enterprise Java Bean (EJB)~~ ENTERPRISE JAVA BEAN® (EJB®) container 106 to facilitate access to EIS and/or database systems 104. A container is a controlled run-time environment for an application component (e.g., an EJB® 108) that also provides an interface to components executing within the container. Also, containers provide basic management functionality such as life cycle management of the components, security, deployment, threading, etc. An EJB® 108 is a software component that implements the interfaces defined by its respective container. The interfaces of the EJB® 108 are defined in a manner that permits the functionality of the respective EJB® 108 to be accessed in a modular and dynamic manner.

Please replace the paragraph at page 8, line 12 to page 9, line 3 with the following amended paragraph:

An exemplary flow diagram of a process 200 for generating code according to the present invention is shown in Fig. 2. Typically, a bean archive file is deployed to an

application server. For example, if the bean archive file contains an ~~Enterprise Java Bean~~ ENTERPRISE JAVA BEAN® (EJB®) implementation, the archive file includes one or more classes of objects. Process 200 begins with step 202, in which the input class, that is the object class included in the archive file, is introspected. Introspection involves extracting information relating to the objects, methods, etc., that are included in the class. For example, information identifying methods included in the input class may be extracted and for each method, information relating to parameters of the method are extracted. In [[Java]] JAVA®, introspection may be performed using the [[Java]] JAVA® Reflection API.

Please replace the paragraph at page 9, lines 4-6 with the following amended paragraph:

In step 204, an XML description of the [[Java]] JAVA® bean class is generated based on the class information gathered via the introspection performed in step 202. For example,

Please replace the paragraph at page 10, lines 5-11 with the following amended paragraph:

In step 208, the event handlers are registered with a SAX parser and, in step 210, the parsing of the [[Java]] JAVA® bean class XML descriptor commences. As the event handlers are invoked by the parser, step 212, the code generation proceeds. Note that this facility could potentially be leveraged to generate more than one file in parallel if, for

example, an event handler leverages multiple templates. For example, if a method affects multiple files, all handlers for a method may be invoked at once and the code for each file generated in parallel.

Please replace the paragraph at page 14, line 18 to page 15, line 2 with the following amended paragraph:

SAX is an event-driven, serial-access mechanism for accessing XML documents. SAX was the first widely adopted API for XML in [[Java]] JAVA®, and is a "de facto" standard. A typical version of SAX contains a number of core classes and interfaces together with optional helper classes and demonstration classes.

Please replace the paragraph at page 19, line 5 with the following amended paragraph:

- Optional [[Java]] JAVA®-specific helper classes:

Please replace the paragraph at page 19, line 8 with the following amended paragraph:

- [[Java]] JAVA® demonstration classes in the nul package:

Please replace the paragraph at page 22, line 16 with the following amended paragraph:

[[Java]] JAVA®-Specific Helper Classes

Please replace the paragraph at page 22, lines 17-19 with the following amended paragraph:

These classes are not part of the core SAX distribution, and may not be available in SAX implementations in other languages: they are provided simply as a convenience for [[Java]] JAVA® programmers.

Please replace the paragraph at page 16, lines 5-9 with the following amended paragraph:

The `SAXParser` [[404]] 402 wraps a `SAXReader` [[402]] 404. Typically, you don't care about that, but every once in a while you need to get hold of it using `SAXParser`'s `getXMLReader()`, so you can configure it. It is the `SAXReader` [[402]] 404 that carries on the conversation with the SAX event handlers you define.

Please replace the Abstract with the amended Abstract submitted herewith on a separate sheet.